First Hit Fwd Refs

Generate Collection Print

L49: Entry 2 of 36 File: USPT Sep 3, 2002

DOCUMENT-IDENTIFIER: US 6446149 B1
TITLE: Self-modifying synchronization memory address space and protocol for communication between multiple busmasters of a computer system

Abstract Text (1):
A computer system provides a self-modifying synchronization memory address space and protocol for communication between multiple busmasters. In one computer system embodiment, the self-modifying synchronization memory address space is provided in a memory controller embedded in a peripheral device of the computer system such as a bridge that provides central, high speed access by a busmaster to the memory controller without accessing a host bus. The synchronization memory address space includes a set of semaphore memory cells mapped to shared critical resources in the computer system. The semaphore memory cell allows for exclusive access by a busmaster to a shared critical resource by switching itself from an idle state to a busy state responsive to a first read operation by a busmaster. In the busy state of the semaphore memory cell, a busy state is communicated to other busmasters which attempt to read the semaphore memory cell. Ownership of the semaphore memory cell is thus achieved using a single operation by a busmaster. The properties of the self-modifying synchronization memory address space and the semaphore memory cell thus eliminate the need for assertion of a bus locking signal to achieve exclusive access for a busmaster to a shared critical resource. These properties also eliminate the need for host processor intervention in accessing a shared critical resource when a busmaster is a PCI master.

Brief Summary Text (3):
The present invention relates to synchronization between multiple busmasters in a computer system, and more particularly to a self-modifying synchronization memory address space and protocol for communication between multiple busmasters.

Brief Summary Text (5):
For computer systems including heterogeneous busmasters and loosely coupled busses, such as distributed computer systems and certain multiprocessor computer systems, shared memory synchronization between multiple busmasters is a primary concern. While homogenous busmasters are generally able to communicate directly with one another, synchronization between heterogenous busmasters and between certain homogenous busmasters, which are unable to communicate directly with one another, has typically required host processor intervention. Host processors have allowed for exclusive accesses by heterogenous busmasters to shared memory by issuing atomic transactions. An atomic transaction is a transaction including operations which are performed without interference from other processes. Atomic transactions have typically included assertion of a bus locking signal. An atomic transaction for achieving an exclusive access for a busmaster has included at least two operations, a read operation and a write or set operation. A semaphore memory variable is a flag object for coordinating exclusive access to a critical region of a shared memory. During a read operation, a semaphore memory variable is read, and during a write or set operation, a value is written to the semaphore memory variable. Implementing atomic transactions has required special processor instructions for performing a sequence of real and write operations. Examples of special processor instructions include `swap` instructions, memory exchange instructions, `test & set` instructions, and `read-modify-write` instructions. The

most prevalent special processor instruction has been the `read-modify-write` instruction. Each special processor instruction, like the `read-modify-write` instruction, essentially performs a read-modify-write cycle. Also, in implementing atomic transactions, it has been necessary to launch an atomic transaction on a bus having a bus protocol supporting the particular special processor instruction.

Brief Summary Text (6):
An algorithm including a special processor instruction for performing a read-modify-write cycle, such as the ACQUIRE_SPINLOCK routine provided in Windows NT.RTM., is used to synchronize access to a shared memory between multiple busmasters. The routine is initiated by a busmaster seeking to determine if a semaphore memory variable may be claimed. The special processor instruction commonly used in the ACQUIRE_SPINLOCK routine to perform a read-modify-write cycle is the Exchange instruction. The Exchange operation is an intrusive operation which includes an assertion of a bus locking signal by a processor.

Brief Summary Text (7):
A bus locking signal such as Intel's LOCK# signal on the X86 processors is typically used in connection with an atomic transaction to permit a busmaster to attempt to claim a semaphore memory variable. When a bus locking signal is asserted, the associated bus is locked, preventing other busmasters from acquiring ownership of the bus and, thus, access to the semaphore memory variable. The locked cycle of the busmaster terminates when the bus locking signal is deasserted. A bus locking signal undesirably consumes or narrows bandwidth of the associated bus. A bus locking signal also has the drawback of being specific for a particular bus locking architecture. For example, a PCI master is presently unable to initiate a LOCK# signal onto a host bus. A further disadvantage is that assertion of a bus locking signal forces posted writes within a host processor to be flushed to main memory.

Brief Summary Text (9):
Briefly, the system and method according to the present invention provides a synchronization memory address space and synchronization memory protocol for communication between multiple busmasters in a computer system. The self-modifying synchronization memory address space is preferably located in a memory controller embedded in a peripheral device such as a bridge that provides central, high speed access by a busmaster to the memory controller without accessing a host bus. The address space includes a set of semaphore memory cells mapped to shared critical resources in the computer system. Each region of the self-modifying synchronization address space corresponding to a particular shared critical resource serves as a synchronization memory channel.

Brief Summary Text (12):
Each bridge in the computer system preferably includes a synchronization bridge configuration register. A synchronization bridge configuration register allows a bridge to determine whether a target address provided by a busmaster is mapped to the self-modifying synchronization memory address space, whether the synchronization memory channel local to the bridge is enabled, and whether a bridge is reflector enabled so as to allow the bridge to reflect a write to another bridge. The synchronization bridge also may indicate the location of the synchronization memory channel local to the bridge.

Detailed Description Text (4):
The memory controllers 30A and 30B include synchronization memory address space regions 28A and 28B. These regions collectively form the self-modifying synchronization memory address space 28 of the present invention. Each region of the synchronization memory address space 28 is preferably located in a memory controller embedded in a peripheral device such as a bridge that provides central, high speed access by a busmaster to the memory controller without accessing the host bus.

Detailed Description Text (5):
Critical regions of the shared host memory 18 and critical devices within the computer system S are mapped into the self-modifying synchronization memory address space 28, which serves as a surrogate target of a request by a busmaster. A region of the address space 28 corresponding to a particular shared resource may be termed a synchronization memory channel. The synchronization memory address space 28 includes a set of semaphore memory cells 60. Each critical region within the shared host memory 18 and each critical device is preferably associated with one of the semaphore memory cells 60. The particular state or mode of a particular semaphore memory cell 60 is communicated to a busmaster requesting the semaphore memory cell 60. It should be understood that the self-modifying synchronization memory address space 28 may be configured as a single synchronization memory channel or a plurality of synchronization memory channels.

Detailed Description Text (6):
For conventional synchronization memory protocols, it has been necessary for a busmaster such as a CPU to assert a LOCK# signal or an atomic transaction to attempt to claim a semaphore associated with a shared critical resource. A LOCK# signal is in a deasserted state when the LOCK# signal is not being used. When a LOCK# signal is asserted during access to a semaphore, another busmaster can not acquire ownership of the semaphore. If a busmaster asserts a LOCK# signal, a bus arbiter locks the associated bus. A LOCK# signal thereby prevents another busmaster from claiming a semaphore from the time the busmaster initiates a read of the semaphore to the time the busmaster writes a value, typically a non-zero value, to the semaphore indicating the semaphore is available. Assertion of the LOCK# signal over a host and PCI bus has been somewhat inefficient. A LOCK# signal undesirably consumes or narrows bandwidth of a PCI bus and host bus. Further under the current PCI specification, a PCI master is unable to initiate a LOCK# signal into host memory; therefore, host processor intervention has been necessary to initiate a LOCK# signal in such circumstances.

Detailed Description Text (7):
The self-modifying synchronization memory address space 28, however, achieves a multi-master synchronization memory protocol without consuming bandwidth of a host bus. By locating the synchronization address space 28 between a host bus and a plurality of busmasters, a busmaster may access the self-modifying synchronization address space 28 without an access of the host bus. The semaphore memory cell 60 of the synchronization address space 28 has the ability to switch itself from an idle state to a busy state in response to a first read operation by a busmaster. In this way, exclusive access to a shared critical resource such as the shared host memory 18 is achieved without assertion of a LOCK# signal. A further advantage when the requesting busmaster is a PCI master is that access to a shared central resource is obtained without host processor intervention. The multi-master synchronization memory protocol is understood by the device drivers associated with busmasters of the computer system S.

Detailed Description Text (8):
Referring to FIG. 2, the self-modifying synchronization memory address space 28 is shown in a physical memory address space 58 of the computer system C. The physical memory address space 58 of a computer system is typically four gigabytes (GB). Within that four gigabyte memory address space, the synchronization memory address space 28 is programmed. The self-modifying synchronization memory address space 28 for example may be a one megabyte (MB) memory "hole" of the physical memory address space 58. In accordance with convention, the bottom of the self-modifying synchronization memory address space 28 may be termed the physical base address of the self-modifying synchronization memory address space 28. Critical regions of the shared host memory 18 and other shared critical resources in the computer system C may be mapped into the self-modifying synchronization memory address space 28.

Detailed Description Text (9):
Referring to FIG. 3, an illustration of the self-modifying semaphore memory address space 28 is shown. Each semaphore memory cell 60 of the semaphore memory address space 28 is preferably a separate cache line or is the size of a cache line in order to exploit concurrency. Since each semaphore 60 guards a mutually exclusive memory region or resource, a busmaster claiming one semaphore 60 should not be able to affect or block a busmaster claiming a different semaphore. If more than one semaphore 60 is provided on a single cache line, a busmaster claiming one semaphore 60 affects a busmaster claiming the other semaphore 60 on the same cache line. In conventional synchronization memory protocols, semaphores have shared cache lines and also have straddled cache lines. A semaphore straddles a cache line where one portion of a semaphore is on one cache line and another portion is on a different cache line. The synchronization memory protocol of the present invention preferably avoids both straddling of a cache line and interleaving of semaphores within a single cache line.

Detailed Description Text (10):
Referring to FIG. 4, a state diagram of a prior art synchronization memory protocol is shown. When a busmaster seeks to claim a semaphore in an idle state 62, the busmaster first asserts a LOCK# signal as indicated in step 66. Next, the busmaster performs an atomic transaction including a series of atomic operations to claim the semaphore. It should be understood that the LOCK# signal may be asserted as part of the atomic transaction or before the atomic transaction. In step 68, the read operation to the semaphore 60 is atomically performed. In this operation, the idle state 62 of the semaphore is communicated to the busmaster. Next, in step 70, the busmaster performs a modify operation internal to the busmaster. Control then passes to step 72 wherein the busmaster performs a write operation to the semaphore to place the semaphore in a busy state. The read-modify-write cycle performed by steps 68-72 includes at least two operations to be performed by a busmaster such as a microprocessor, namely, a read operation and a write operation. If a semaphore is split across address boundaries, cache line boundaries, or page boundaries, more than two transactions have historically been necessary to accomplish a read-modify-write cycle. If a semaphore is in a busy state 64, 7a write operation by the busmaster to the semaphore as indicated in step 74 returns the semaphore to an idle state 62. ACQUIRE_SPINLOCK is a routine provided in Windows NT.RTM. to synchronize access to a shared critical resource such as a shared region in host memory between multiple busmasters. The algorithm is initiated by a busmaster seeking to determine if a semaphore may be claimed. An example of an ACQUIRE_SPINLOCK algorithm in assembly language pursuant to the conventional synchronization memory protocol illustrated in FIG. 4 follows:

Detailed Description Text (11):
In the first line of the code, the number `1` is stored in an EAX register by a move (MOV) instruction. The EAX register is a 16 bit general purpose register of a microprocessor when that processor is a x86 compatible processor. Next, the semaphore is compared to a `0` through a compare (CMP) instruction. For this particular memory protocol, a `1` represents a busy state of a semaphore and a `0` represents an idle state of a semaphore. Next, a jump not equal (JNE) instruction is executed. If the semaphore is not equal to `0,` then the jump not equal operation jumps back to the compare instruction termed Spin, and the compare instruction is again executed to compare the semaphore to a `0`. Following execution of the compare instruction, the jump not equal instruction is again executed. As long as the semaphore is not equal to `0`, the code continues to loop back to the compare instruction. If the semaphore is not equal to `0`, the semaphore is equal to a `1` indicating to a busmaster that the semaphore is in a busy state. If the jump not equal instruction detects that the semaphore is equal to `0`, indicating the semaphore requested by a busmaster is in an idle state, the code proceeds to the next instruction.

Detailed Description Text (12):

h e b b g e e e f c e b e ge

The code next executes the Exchange instruction (XCHG) performing an exchange operation in which the contents of the semaphore and the contents of the EAX register are exchanged or swapped. If the Exchange operation was successful, then the EAX register contains a `0`. The Exchange instruction includes an assertion of a LOCK# signal by a processor and is therefore an intrusive operation. It should be understood that certain instructions, if prefixed with a lock prefix, may force an assertion of a LOCK# signal. As noted earlier, a LOCK# signal requires host processor intervention and also consumes or narrows bandwidth of a host bus and PCI bus. The exchange operation of the ACQUIRE_SPINLOCK algorithm corresponds to the atomic read-modify-write cycle and LOCK# signal assertion of FIG. 4. An atomic read-modify-write cycle using a LOCK# signal causes bus thrashing and flushing of write-posting buffers of a host processor. If a semaphore is cacheable, an atomic read-modify-write operation may also be performed by locking the semaphore cache line. Locking a semaphore cache line has also required assertion of a LOCK# signal.

Detailed Description Text (13):
Following execution of the Exchange instruction, a compare instruction for comparing the EAX register to a `0` is executed. Since the contents of the semaphore was placed in the EAX register by the Exchange instruction, the EAX register will contain a `0` if the semaphore contained a `0`. Next, a jump not equal instruction is executed such that the code jump backs to the compare instruction designated Spin if the EAX register does not contain a `0`. If the EAX register does not contain a `0`, the EAX register contains a `1` indicating the semaphore is in a busy state. If the semaphore is in a busy state, another busmaster has claimed the semaphore ahead of the LOCK# signal asserted by the busmaster seeking to claim the semaphore. If the requesting busmaster is cacheable, the compare or Spin operation is directed to the semaphore in the cache of the requesting busmaster. If the EAX register contains a `0`, the semaphore is in an idle state and may be claimed by the busmaster initiating the ACQUIRE_SPINLOCK algorithm.

Detailed Description Text (14):
Referring to FIG. 5, a state diagram of the synchronization memory protocol in a system according to the present invention is shown. If a busmaster issues a read operation in step 80 to a semaphore 60 in an idle state 76, the semaphore memory cell 60 switches itself to a busy state 78. Any subsequent reads by another busmaster will thus read busy. Unlike conventional synchronization memory protocols, a busmaster only performs a single operation, being the read operation of step 80 to attempt to claim a semaphore 60. In addition, it is unnecessary for a busmaster to comprehend a LOCK# signal since the semaphore memory cell 60 switches itself to a busy state. A busmaster therefore instead need only comprehend a basic read operation to detect if a semaphore memory cell 60 is in a busy state or an idle state. The read operation illustrated in step 80 corresponds to a first read by a busmaster.

Detailed Description Text (16):
The improved ACQUIRE_SPINLOCK algorithm includes only two assembly language instructions. The first instruction is a compare instruction for comparing the semaphore 60 to a `0`. Control then proceeds to execution of a jump not equal instruction. The jump not equal instruction detects whether the semaphore 60 is not equal to `0`. If the semaphore 60 is not equal to `0`, the code loops back to the compare instruction which is again executed. As long as the semaphore 60 is not equal to `0`, the compare instruction and jump not equal instruction are repeatedly executed. If the jump not equal instruction detects that the semaphore 60 is equal to `0`, indicating the semaphore 60 is in an idle state, the code terminates. The improved ACQUIRE_SPINLOCK algorithm does not require assertion of a LOCK# signal via an exchange instruction because the semaphore 60 switches itself to a busy state.

Detailed Description Text (24):
Referring to FIGS. 8A-8E, a synchronization memory protocol according to the present invention is illustrated. In FIG. 8A, CPU-110 performs a read operation to the semaphore memory cell 60 or semaphore memory bit 84. The `0` or idle state of the semaphore memory 60 is thereby communicated to CPU-110. Next, in FIG. 8B, the semaphore memory cell 60 switches itself from a `0` or idle state to a `1` or busy state. In a conventional synchronization memory protocol, it has been necessary not only for the CPU to write a `1`, or busy state, to a semaphore, but also to lock the associated bus during that process. A conventional synchronization protocol therefore has required an atomic transaction including at least two atomic operations, a read operation and a write operation, on the part of a busmaster. In contrast, according to the present invention, providing a semaphore memory cell 60 capable of switching itself from an idle state to a busy state reduces the number of operations performed needed by a busmaster. The synchronization protocol of the present invention in effect allows for an exclusive read mode by a busmaster. Therefore, use of the semaphore memory cell 60 eliminates the need to assert a LOCK# signal for semaphore operations.

Detailed Description Text (26):
Referring to FIG. 9, an illustration of a definition for a synchronization address 86 provided by a requesting busmaster to address a semaphore 60 in the self-modifying synchronization memory address space 28 is shown. Bits 20-31 of the synchronization address 86 represent the physical base address for the self-modifying synchronization memory address space 28. Bits 5-19 represents the particular address of a semaphore 60 requested by a busmaster. Bits 3-4 represent an encoded identification value for a PCI master. This set of bits is only applicable for a request by a PCI master. The PCI master identification value is used to inform a bridge of which PCI master is requesting a semaphore. Bits 0-2 (not shown) are undefiled for the synchronization address 86. This exemplary definition of a synchronization address 86 provides for 32 bits. It should be understood that the number of bits for definition of the synchronization address 86 may vary.

Detailed Description Text (34):
Referring to FIG. 14, a state diagram for the states of a cache of a cacheable busmaster in accordance with the synchronization memory protocol of the present invention is shown. In accordance with MESI protocol, the standard protocol for cache line states, the four possible states for cache line are represented. The M state 104 refers to a modified cache line; the S state 106 refers to a shared cache line; the E state 110 refers to an exclusive cache line; and the I state 108 refers to an invalid cache line. In accordance with the synchronization memory protocol of the present invention, certain cache line state transitions controlled by a cache controller are used to preserve cache consistency. If a write operation is performed by a busmaster to one of the semaphore memory cells 60, the semaphore cache line corresponding to the semaphore memory cells 60 of a cacheable busmaster receiving the write is marked invalid, thereby placing the cache line in an invalid state. W.sub.1 represents a write operation 112 triggering a state transition from a semaphore cache line 160 in a modified state 104 to an invalid state 108. W.sub.2 represents a write operation 120 triggering a state transition from a semaphore cache line 60 in an exclusive state 110 to an invalid state 108. W.sub.3 represents a write operation 116 triggering a state transition from a semaphore cache line 60 in a shared state 106 to an invalid state 108. W.sub.4 represents a write operation 113 maintaining an invalid state for a semaphore cache line 60.

Detailed Description Text (35):
Lastly, the synchronization memory protocol according to the present invention is particular useful for a computer system C including heterogenous busmasters and loosely coupled busses. In a computer system C including heterogenous busmasters and loosely coupled busses, busmasters are unable to communicate with each other without host processor intervention. A system according to the present invention

h e b b g e e e f c e b e ge

eliminates the need for host processor intervention for certain transactions by providing a self-modifying synchronization memory address space allowing heterogeneous busmasters to directly communicate with one another. As a result, the amount of external interrupt activity generated back to host processors in a computer system is reduced in accordance with the present invention. For example, one heterogeneous busmaster may synchronize itself with another heterogenous busmaster by requesting a semaphore in the self-modifying synchronization memory address space associated with the other busmaster.

US Reference Patentee Name (7):
Lockwood

US Reference Group (7):
5339443 19940800 Lockwood 395/725

CLAIMS:

1. A computer system adapted for a self-modifying synchronization memory protocol for multiple busmasters, comprising: at least one host processor; at least one host bus coupled to the host processor; a plurality of busmasters; a self-modifying synchronization memory address space coupled to the host bus and including at least one semaphore memory cell having an idle state or a busy state, the self-modifying synchronization memory address space being coupled between the host bus and the plurality of busmasters, the self-modifying synchronization memory address space being accessible by the plurality of busmasters independent of the host bus; and the semaphore memory cell switching itself to a busy state only if the semaphore memory cell is read by a busmaster of the plurality of busmasters when the semaphore memory cell is in an idle state so that the semaphore memory cell is owned by the busmaster, wherein the semaphore memory cell retains the busy state until a subsequent action by the busmaster that owns the semaphore memory cell triggers the semaphore memory cell to switch itself to the idle state, and wherein a busmaster desiring ownership of the semaphore memory cell repeatedly reads the semaphore memory cell until is owned by the busmaster desiring ownership of the semaphore memory cell.

3. The computer system of claim 1, further comprising: at least one shared resource mapped into the self-modifying synchronization memory address space.

4. The computer system of claim 1, the computer system including a memory controller, wherein the self-modifying synchronization address space is located in the memory controller.

6. The computer system of claim 1, wherein the self-modifying synchronization address space is a non-cacheable address space.

7. The computer system of claim 1, wherein the self-modifying synchronization address space is a write-through cacheable address space.

8. The computer system of claim 1, wherein the self-modifying synchronization address space is a writeback-invalidate cacheable address space.

17. The computer system of claim 1, the computer system including a bridge coupled to the host bus, further comprising: synchronization bridge configuration logic provided in the bridge for configuring the self-modifying synchronization memory address space.

18. The computer system of claim 17, the synchronization bridge configuration logic comprising: a synchronization memory address region enable bit for selectively enabling a region of the self-modifying synchronization memory address space in the bridge.

h e b b g e e e f c e b e ge

21. The computer system of claim 20, the synchronization address comprising: a set of bits representing the physical address of the self-modifying synchronization memory address space; a set of bits representing an address of a semaphore memory cell of the self-modifying synchronization memory address space; and a set of bits representing a PCI busmaster identification value to inform a bridge of the identity of a PCI busmaster issuing a request to a semaphore memory cell.

22. The computer system of claim 1, further comprising: a synchronization memory bus for providing a semaphore address to the self-modifying synchronization memory address space and providing a semaphore memory cell from the self-modifying synchronization memory address space.

29. A synchronization method permitting shared memory communication between multiple busmasters in a computer system using semaphore memory cells of a synchronization memory address space, each semaphore memory cell having an idle state or a busy state, comprising the steps of: reading state information in a semaphore memory cell by a busmaster, the semaphore memory cell being coupled between a host bus of the computer system and the busmaster, the self-modifying synchronization memory address space being accessible by the plurality of busmasters independent of the host bus; self-switching of the semaphore memory cell to a busy state responsive to said read to the semaphore memory cell if the state information read by the busmaster corresponds to an idle state so that the busmaster owns the semaphore memory cell; and retaining the busy state of the semaphore memory cell until a subsequent action by the busmaster that owns the semaphore memory cell triggers the semaphore memory cell to switch itself to the idle state, wherein the step of reading state information is performed repeatedly by the busmaster until the semaphore memory cell is in the idle state.

30. The synchronization method of claim 29, further comprising the steps of writing state information to the semaphore memory cell by the busmaster owning the semaphore memory cell; and self-switching of the semaphore memory cell to an idle state responsive to said write to the semaphore memory cell.

38. A synchronization method permitting shared memory communication between multiple busmasters in a computer system using semaphore memory cells of a self-modifying synchronization memory address space, comprising the steps of: reading state information in a semaphore memory cell by a busmaster to determine if the semaphore memory cell is owned by another busmaster, the semaphore memory cell being coupled between a host bus of the computer system and the busmaster, the self-modifying synchronization memory address space being accessible by the plurality of busmasters independent of the host bus; and repeatedly reading state information in the semaphore memory cell by the busmaster until the semaphore memory cell is owned by the busmaster.

40. The synchronization method of claim 38, further comprising the step of: self-switching of the semaphore memory cell to an idle state responsive to a write to the semaphore memory cell by the busmaster owning the semaphore memory cell so that the busmaster releases ownership of the semaphore memory cell.

42. The synchronization method of claim 38, further comprising the step of: self-switching of the semaphore memory cell to a busy state responsive to a first read to the semaphore memory cell after a write to the semaphore memory cell.

43. A computer system adapted for a self-modifying synchronization memory protocol for multiple busmasters, comprising: at least one processor; at least one host bus coupled to the host processor; a plurality of busmasters; a self-modifying synchronization memory address space coupled to the host bus and including at least one semaphore memory cell having an idle state or a busy state, the self-modifying synchronization memory address space being coupled between the host bus and the

h e b b g e e e f c e b e ge

plurality of busmasters, the self-modifying synchronization memory address space being accessible by the plurality of busmasters independent of the host bus; the semaphore memory cell switching itself to a busy state only if the semaphore memory cell is read by a busmaster of the plurality of busmasters when the semaphore memory cell is in an idle state so that the semaphore memory cell is owned by the busmaster; and a shared critical resource mapped into the self-modifying synchronization memory address space, wherein the semaphore memory cell retains the busy state until a subsequent action by the busmaster that owns the semaphore memory cell triggers the semaphore memory cell to switch itself to the idle state, and wherein a busmaster desiring ownership of the semaphore memory cell repeatedly reads the semaphore memory cell until the semaphore memory cell is owned by the busmaster desiring ownership of the semaphore memory cell.

45. The computer system of claim 43, the computer system including a memory controller, wherein the self-modifying synchronization address space is located in the memory controller.